UG_N32G032 算法库使用指南

V1.0.0

国民技术股份有限公司 Nations Technologies Inc.
地址：深圳市南山区高新北区宝深路 109 号国民技术大厦
电话：+86-755-86309900 传真：+86-755-86169100
邮箱：info@nationstech.com 邮编：518057

声明

国民技术股份有限公司（下称“国民技术”）对此文档拥有专属产权。依据中华人民共和国的法律、条约以及世界其他法域相适用的管辖，此文档及其中描述的国民技术产品（下称“产品”）为公司所有。

国民技术在此并未授予专利权、著作权、商标权或其他任何知识产权许可。所提到或引用的第三方名称或品牌（如有）仅用作区别之目的。

国民技术保留随时变更、订正、增强、修改和改良此文档的权利，恕不另行通知。请使用人在下单购买前联系国民技术获取此文档的最新版本。

国民技术竭力提供准确可信的资讯，但即便如此，并不推定国民技术对此文档准确性和可靠性承担责任。

使用此文档信息以及生成产品时，使用者应当进行合理的设计、编程并测试其功能性和安全性，国民技术不对任何因使用此文档或本产品而产生的任何直接、间接、意外、特殊、惩罚性或衍生性损害结果承担责任。

国民技术对于产品在系统或设备中的应用效果没有任何故意或保证，如有任何应用在其发生操作不当或故障情况下，有可能致使人员伤亡、人身伤害或严重财产损失，则此类应用被视为“不安全使用”。

不安全使用包括但不限于：外科手术设备、原子能控制仪器、飞机或宇宙飞船仪器、所有类型的安全装置以及其他旨在支持或维持生命的应用。

所有不安全使用的风险应由使用人承担，同时使用人应使国民技术免于因为这类不安全使用而导致被诉、支付费用、发生损害或承担责任时的赔偿。

对于此文档和产品的任何明示、默示之保证，包括但不限于适销性、特定用途适用性和不侵权的保证责任，国民技术可在法律允许范围内进行免责。

未经明确许可，任何人不得以任何理由对此文档的全部或部分进行使用、复制、修改、抄录和传播。

注 意

这是国民技术不便于披露的文件，它包含一些保密的信息。在没有签订任何保密协议前或者在国民技术单方面要求的情况下请归还于国民技术。任何非国民技术委托人不得使用或者参考该文件。

如果你得到了这份文件，请注意：

- 不得公开文档内容
- 不得转载全部或部分文档内容
- 不得修改全部或部分文档内容

在以下情况这份文件必须销毁

- 国民技术已经提供更新的版本
- 未签订保密协议或者保密协议已经过期
- 受委托人离职

给我们的客户

我们一直在不断的改进我们的产品及说明文档的品质。我们努力保证这份文档的说明是准确的，但也可能存在一些我们未曾发现的失误。如果您发现了文档中有任何疑问或错失的地方请及时联系我们。您的理解及支持将使得这份文档更加完善。

版本历史

版本	日期	备注
V1.0	2020.07.16	新建文档
		。

术语及缩略语

缩写	全拼
AES	Advance Encryption Standard
RNG	Random Number Generator

目 录

1. 概述

本文档适用于已下载相关算法的 N32G032 芯片，主要说明该类芯片中算法接口和使用方法。对于 U32 数据类型参数，若采用 U8 强制转换 U32 形式，则需要确保 U8 地址按字对齐。

1.1. 支持的算法

N32G032 芯片提供的算法如下：

□ AES: 加密/解密（AES-128/192/256）

□ SM4: 加密/解密

□ RNG: 随机数生成

1.2. 基本数据类型

```
typedef unsigned char              bool;
typedef unsigned char              u8;
typedef signed char                s8;
typedef unsigned short             u16;
typedef signed short              s16;
typedef unsigned int              u32;
typedef signed int                 s32;
typedef unsigned long long         u64;
typedef signed long long          s64;
```

2. AES算法API说明

2.1. 算法库使用方法

算法库使用方法如下：

1. 将n32g032_aes.h、n32g032_algo_common.h中；将n32g032_algo_common.lib、n32g032_aes.lib程中；
2. 按2.3节函数说明调用函数，例程见附录一提供的demo

2.2. 数据类型定义

```
  #define AES_ECB (0x11111111)
#define AES_CBC (0x22222222)
#define AES_CTR (0x33333333)

#define AES_ENC    (0x44444444)
#define AES_DEC    (0x55555555)
enum
{
      AES_Crypto_OK = 0x0,      //AES opreation success
      AES_Init_OK = 0x0,      //AES Init opreation success
      AES_Crypto_ModeError = 0x5a5a5a5a,      //Working mode error(Neither ECB nor CBC nor CTR)
      AES_Crypto_EnOrDeError,        //En&De error(Neither encryption nor decryption)
      AES_Crypto_ParaNull,        // the part of input(output/iv) Null
      AES_Crypto_LengthError,          // if Working mode is ECB or CBC,the length of input message must
be 4 times and cannot be zero;
                                                            //if Working mode is CTR,the length of input message cannot be
zero; othets: return AES_Crypto_LengthError
```

```
    AES_Crypto_KeyLengthError, //the keyWordLen must be 4 or 6 or 8; othets:return
    AES_Crypto_KeyLengthError
    AES_Crypto_UnInitError, //AES uninitialized
};
typedef struct
{
        uint32_t *in;       // the part of input to be encrypted or decrypted
        uint32_t *iv;       // the part of initial vector
        uint32_t *out;      // the part of out
        uint32_t *key;      // the part of key
        uint32_t keyWordLen;      // the length(by word) of key
        uint32_t inWordLen; // the length(by word) of plaintext or cipher
        uint32_t En_De; // 0x44444444- encrypt, 0x55555555 - decrypt
        uint32_t Mode;    // 0x11111111 - ECB, 0x22222222 - CBC, 0x33333333 - CTR
}AES_PARM;
```

2.3. 函数接口说明

AES 算法库包含的函数列表如下：

表 2-1　AES 算法库函数表

函数	描述
uint32_t AES_Init(AES_PARM *parm)	AES 初始化
uint32_t AES_Crypto(AES_PARM *parm)	AES 加解密函数
void AES_Close(void)	AES 关闭函数
void AES_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)	AES 版本获取函数

2.3.1. AES算法初始化

AES_Init	AES 算法初始化
函数原型	uint32_t AES_Init(AES_PARM *parm)
参数说明	parm　　输入，指向 AES _PARM 结构体的指针
返回值	AES_Init_OK：运算正确　　其他：运算错误
注意事项	1.调用方式请参考附录一。

2.3.2. AES算法加解密

AES_Crypto	AES 算法加解密
函数原型	uint32_t AES_Crypto(AES_PARM *parm)
参数说明	parm　　输入，指向 AES _PARM 结构体的指针
返回值	AES_Crypto_OK：运算正确　　其他：运算错误
注意事项	在调用本函数前，若还未初始化或已切换到其他算法，先调用 AES_Init 函数； 1.调用方式请参考附录一。

2.3.3. 关闭AES

AES_Close	关闭 AES 算法时钟和系统时钟
函数原型	void AES_Close(void)
参数说明	
返回值	

2.3.4. 获取AES库版本信息

AES_Version	获取 AES 库版本信息

函数原型 void AES_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)

参数说明

type	商业或快速版本
customer	标准或定制版本
date	年，月，日
version	//版本 x.x

返回值

注意事项

*type = 0x05; // 商业和快速版

*customer = 0x00; // 标准版本

date[0] = 20; //Year()

date[1] = 7; //Month()

date[2] = 16; //Day ()

*version = 0x10; //表示版本 1.0

3. SM4算法API说明

3.1. 算法库使用方法

算法库使用方法如下：

1. 将 n32g032_sm4.h、n32g032_algo_common.h 加入头文件夹中，将 n32g032_algo_common.lib、n32g032_sm4.lib 添加到工程中；
2. 按 3.3 节函数说明调用函数，例程见附录二提供的 demo

3.2. 数据类型定义

```
#define SM4_ECB (0x11111111)
#define SM4_CBC (0x22222222)
#define SM4_ENC    (0x33333333)
#define SM4_DEC    (0x44444444)
enum{
                SM4_Crypto_OK=0, //SM4 opreation success
               SM4_Init_OK=0, //SM4 Init opreation success
              SM4_ADRNULL =0x27A90E35, //the address is NULL
              SM4_ModeErr, //working mode error(Neither ECB nor CBC)
              SM4_EnDeErr,    // En&De error(Neither encryption nor decryption)
              SM4_LengthErr,//the word length of input error(the word length is 0 or is not as times as 4)
             SM4_UnInitError,      //SM4 uninitialized
};


typedef struct{
      uint32_t *in;      // the first part of input to be encrypted or decrypted
      uint32_t *iv;      // the first part of initial vector
```

```
    uint32_t *out;    // the first part of out
    uint32_t *key;    // the first part of key
    uint32_t inWordLen; //the word length of input or output
    uint32_t EnDeMode; //encrypt/decrypt
    uint32_t workingMode; //    ECB/CBC
}SM4_PARM;
```

3.3. 函数接口说明

SM4 算法库包含的函数列表如下：

表 3-1　SM4 算法库函数表

函数	描述
uint32_t　SM4_Init(SM4_PARM *parm)	SM4 算法初始化函数
uint32_t　SM4_Crypto(SM4_PARM *parm)	SM4 算法加密/解密
void SM4_Close(void)	SM4 算法关闭
void SM4_Version(*uint8_t* *type, *uint8_t* *customer, *uint8_t* date[3], *uint8_t* *version)	获取 SM4 库版本信息

3.3.1.SM4模块初始化

SM4_Init　　初始化 SM4 模块

函数原型　　*uint32_t*　SM4_Init(SM4_PARM *parm)

参数说明　　parm　输入，指向 SM4_PARM 结构体的指针

返回值　　SM4_Init_OK：运算正确　　其他值：计算错误，详见枚举类型值定义

注意事项

3.3.2. SM4算法加解密

SM4_Crypto　　SM4 模块算法加解密

函数原型　　*uint32_t*　SM4_Crypto(SM4_PARM *parm)

参数说明　　parm　输入，指向 SM4_PARM 结构体的指针

返回值　　SM4_Crypto_OK：运算正确　　其他值：计算错误，详见枚举类型值定义

注意事项　　在调用本函数前，若还未初始化或已切换到其他算法，先调用 SM4_Init 函数；

1. 结构体 SM4_PARM 参考 6.2 节 *SM4_PARM 的定义。*

2.若是 ECB 模式，则参数 iv1 可直接用 NULL 替换

3. 大量数据作为一整体但分多块进行 CBC 加密时，需注意：

第 X 块数据（X>1）调用本函数进行加密，使用的初始向量 IV（IV = iv1）一定要更新为第 X-1 块数据调用本函数进行加密得到的密文的最后一个分组（16 字节）。

4. 大量数据作为一整体但分多块进行 CBC 解密时，需注意：

第 X 块数据（X>1）调用本函数进行解密，使用的初始向量 IV（IV = iv1）一定要更新为第 X-1 块数据的最后一个分组（16 字节）

3.3.3. SM4关闭

SM4_Close　　关闭 SM4 算法时钟和系统时钟

函数原型　　void SM4_Close(void)

参数说明

返回值

注意事项

3.3.4. 获取SM4库版本信息

SM4_Version　　获取 SM4 库版本信息

函数原型 void SM4_Version(uint8_t *type, uint8_t *customer, uint8_t date[3], uint8_t *version)

参数说明

type	商业或快速版本
customer	标准或定制版本
date	年，月，日
version	//版本 x.x

返回值

注意事项

*type = 0x05; // 商业和快速版

*customer = 0x00; // 标准版本

date[0] = 20; //Year()

date[1] = 4; //Month()

date[2] = 8; //Day ()

*version = 0x11; //表示版本 1.1

4. RNG算法API说明

4.1. 算法库使用方法

算法库使用方法如下：

1、将 n32g032_rng.h、n32g032_algo_common.h 加入头文件夹中，将 n32g032_algo_common.lib 、n32g032_rng.lib 添加到工程中；

2、按 4.3 节函数说明调用函数。

4.2. 数据类型定义

```
enum{
RNG_OK = 0x5a5a5a5a,
    LENError = 0x311ECF50,     //RNG generation of key length error
     ADDRNULL = 0x7A9DB86C,      // This address is empty
};
```

4.3. 函数接口说明

RNG 算法库包含的函数列表如下：

表 7-1　RNG 算法库函数表

函数	描述
uint32_t GetPseudoRand_U32(*uint32_t* *rand, *uint32_t* wordLen, *uint32_t* seed[2])	伪随机数按 word 生成函数
uint32_t GetTrueRand_U32(*uint32_t* *rand, *uint32_t* wordLen)	真随机数按字生成函数
void RNG_Version(*uint8_t* *type, *uint8_t* *customer, *uint8_t* date[3], *uint8_t* *version)	获取 RNG 库版本信息

4.3.1. 伪随机生成函数

GetPseudoRand_U32 伪随机数按 word 生成函数

函数原型	*uint32_t* GetPseudoRand_U32(*uint32_t* *rand, *uint32_t* wordLen, *uint32_t* seed[2])
参数说明	rand 指针，指向生成的随机数 wordlen: 拟获取伪随机数word长 seed[2] 输入，伪随机种子变量数组
返回值	RNG_OK 成功 ； 其他 生成伪随机数出错
说明	按word生成伪随机数
注意事项	1. 用户可输入种子数组，如果用户输入seed为NULL，则内部自动生成种子；
例程	

4.3.2. 随机数生成函数

GetTrueRand_U32真随机数生成函数

函数原型	*uint32_t* GetTrueRand_U32(*uint32_t* *rand, *uint32_t* wordLen)
参数说明	rand: 指针，指向生成的随机数某内存地址 wordLen: 拟获取真随机数的字长度
返回值	RNG_OK 成功； 其他：生成真随机数出错，详见枚举类型值定义
注意事项	

4.3.3. 获取RNG库版本信息

RNG_Version 获取 RNG 库版本信息

函数原型	void RNG_Version(*uint8_t* *type, *uint8_t* *customer, *uint8_t* date[3], *uint8_t* *version)
参数说明	type 商业或快速版本

customer	标准或定制版本
date	年，月，日
version	//版本 x.x

返回值

注意事项

```
*type = 0x05; //  商业和快速版
*customer = 0x00; //  标准版本
date[0] = 20; //Year()
date[1] = 4; //Month()
date[2] = 8; //Day ()
*version = 0x10;    //表示版本 1.0
```

i.附录一 AES算法库函数调用例程

```
u32 AES_128_test()
{
    u32 flag1,flag2,flag3,flag4,flag5,flag6;
    u32 ret;
    AES_PARM AES_Parm={0};
    /*若需要修改测试实例，当参数的真实值为“0x0102030405060708”时，由于 u32 数据是字节
小端序存储，在对以上参数进行初始化赋值时，请输入“0x04030201,0x08070605”.若无特殊说明，
本例程参数都以这种方式设置*/
    u32 in[32]={0x4A8770A5,0x73C2DA98,0xF52D52D1,0x5F884A46,0x8DCF72D5,0x2A0F207D,
    0x7479F5CE,0x3FB5BE9E,0x3D7998FE,0x7C59586D,0x30E1294B,0xB3E17790,
    0xCA080CBD,0x2AB47913,0x3B09B803,0x1B410FE7,0xE64237EF,0x3576BE5E,
    0xE4D7AAF6,0x19495FB0,0x812DC3B1,0xDD339F7A,0xBE6F495F,0x8CB0803A,
    0xCD0D9760,0xA4C0D6D4,0x98381DBB,0x9769CA10,0x3B67DD99,0x4C335A1A,
    0x85D4EFC8,0x9BAAD700};
    /*in=0xA570874A98DAC273D1522DF5464A885FD572CF8D7D200F2ACEF579749EBEB53FFE9
8793D6D58597C4B29E1309077E1B3BD0C08CA1379B42A03B8093BE70F411BEF3742E65EBE7635
F6AAD7E4B05F4919B1C32D817A9F33DD5F496FBE3A80B08C60970DCDD4D6C0A4BB1D389810
CA699799DD673B1A5A334CC8EFD48500D7AA9B*/

    u32 key[4]={0x7FDDA35D,0x7D5C725B,0x1960F327,0x4FD9DDA2};
    /*key=0x5DA3DD7F5B725C7D27F36019A2DDD94F*/

    u32 iv[4]={0x7B00FE39,0xD3E06638,0xD52BC983,0x38E98017};
     /*iv=0x39FE007B3866E0D383C92BD51780E938*/
    u32 out[32];
    u32 AES_ECB_EN[32]={0xB24E5438,0x0145A303,0xC450A27F,0x2ADEEE70,0x906F314E,
    0xB24229AD,0x1312360E,0x949C8B22,0xE2C1BC02,0x1960239E,
    0xCAD2D5E5,0x8DC57DE2,0x13429CE1,0xE8FC0876,0xCA4581DB,
    0x08019050,0x4B2942F8,0xD6073C62,0x113FB648,0x1967CC27,
    0x250B9989,0x861180E0,0x1A450E0C,0x81D727AF,0xB679608E,
    0x53D31669,0x1D071E99,0x42CEB6DB,0x44094205,0xD0331668,
    0x2704B798,0x6E347E9C};
    /*AES_ECB_EN=0x38544EB203A345017FA250C470EEDE2A4E316F90AD2942B20E361213228
B9C9402BCC1E29E236019E5D5D2CAE27DC58DE19C42137608FCE8DB8145CA50900108F842294
B623C07D648B63F1127CC671989990B25E08011860C0E451AAF27D7818E6079B66916D353991E07
1DDBB6CE4205420944681633D098B704279C7E346E*/

    u32 AES_ECB_DE[32]={0x818D1AFD,0xEC4B4F8E,0x69D9F9FF,0x5567B549,0x42DD5C4B,
    0x3BCA1DD3,0xF318E616,0x89297FEC,0x2A3E0A06,0xFDA90D61,
    0x93DCAE5D,0xCF1AFEAE,0x3CF5A889,0x4CFFEFE3,0xB2C42607,
    0x37D43F8A,0x9C1CD1D8,0x2FE878E8,0x22D941C3,0x239B9D2D,
    0xD9FEB719,0xA4F9E01C,0xC9C39FE8,0x336B01FA,0xFD12E415,
    0x2B6A0006,0x4A35AFBC,0xA7942FAB,0x09DF0A3A,0x9545521B,
    0x7E009336,0x030A5DA5};
    /*AES_ECB_DE=0xFD1A8D818E4F4BECFFF9D96949B567554B5CDD42D31DCA3B16E618F3
EC7F2989060A3E2A610DA9FD5DAEDC93AEFE1ACF89A8F53CE3EFFF4C0726C4B28A3FD437D
8D11C9CE878E82FC341D9222D9D9B2319B7FED91CE0F9A4E89FC3C9FA016B3315E412FD06006
A2BBCAF354AAB2F94A73A0ADF091B5245953693007EA55D0A03*/

    u32 AES_CBC_EN[32]={0x8A83E006,0xAC3AB610,0x0CD2C4CB,0x21F22AA9,0x61963E3C,
    0x992FDE54,0x7E408523,0x749261FF,0xE159802D,0xBC807E3C,
    0x1C16AF67,0xE7574629,0x73573225,0xEE88600D,0x324FE0BB,
    0x7426A48C,0x8EA9E470,0x4DB1BE0F,0x9DC49C2E,0xAD41A05B,
    0x9E7C9143,0x15F55BF2,0xF4E7195D,0x2D9E1E46,0xB78E9809,
    0xF8F831D0,0x12F1890A,0x0CABFF9C,0x49E6FCE6,0x6156CDA5,
    0xFFE38EF7,0x4962AF1D};
    /*AES_CBC_EN=0x06E0838A10B63AACCBC4D20CA92AF2213C3E966154DE2F992385407EF
F6192742D8059E13C7E80BC67AF161C294657E7253257730D6088EEBBE04F328CA4267470E4A98
E0FBEB14D2E9CC49D5BA041AD43917C9EF25BF5155D19E7F4461E9E2D09988EB7D031F8F80A
89F1129CFFAB0CE6FCE649A5CD5661F78EE3FF1DAF6249*/

    u32 AES_CBC_DE[32]={0xFA8DE4C4,0x3FAB29B6,0xBCF2307C,0x6D8E355E,0x085A2CEE,
    0x4808C74B,0x0635B4C7,0xD6A135AA,0xA7F178D3,0xD7A62D1C,
    0xE7A55B93,0xF0AF4030,0x018C3077,0x30A6B78E,0x82250F4C,
    0x8435481A,0x5614DD65,0x055C01FB,0x19D0F9C0,0x38DA92CA,
    0x3FBC80F6,0x918F5E42,0x2D14351E,0x2A225E4A,0x7C3F27A4,
    0xF6599F7C,0xF45AE6E3,0x2B24AF91,0xC4D29D5A,0x318584CF,
    0xE6388E8D,0x946397B5};

    u32 AES_CTR_EN[32]={0xF14C3DA0,0xA74E1089,0x81480939,0x5C8D4E8D,0x655E20AB,
        0x6D797028,0x1E355F48,0x58184929,0x52B1495A,0xC15EB91D,0xFBD499AB,
        0xF59B39FE,0x96DAE1C3,0x6ECC9CDA,0xDA1FB535,0xAA1C74B2,0xA3F19C5E,
      0x9944E1A6,0xDAA05E9A,0xB96278E3,0x1E4915FC,0xB77FBBD2,0x92BA80B9,
      0xCA97857E,0x509D0365,0x78A6FD99,0xB56F5B3C,0xFBEFF5B2,0xF9E928C6,
      0xBC28AE3A,0xD8B82D7A,0xA99BF98D};
    u32
AES_CTR_DE[32]={0x4A8770A5,0x73C2DA98,0xF52D52D1,0x5F884A46,0x8DCF72D5,0x2A0F207
D,
    0x7479F5CE,0x3FB5BE9E,0x3D7998FE,0x7C59586D,0x30E1294B,0xB3E17790,
    0xCA080CBD,0x2AB47913,0x3B09B803,0x1B410FE7,0xE64237EF,0x3576BE5E,
    0xE4D7AAF6,0x19495FB0,0x812DC3B1,0xDD339F7A,0xBE6F495F,0x8CB0803A,
    0xCD0D9760,0xA4C0D6D4,0x98381DBB,0x9769CA10,0x3B67DD99,0x4C335A1A,
    0x85D4EFC8,0x9BAAD700};

    /*AES_CBC_DE=0xC4E48DFAB629AB3F7C30F2BC5E358E6DEE2C5A084BC70848C7B43506
AA35A1D6D378F1A71C2DA6D7935BA5E73040AFF077308C018EB7A6304C0F25821A48358465D
D1456FB015C05C0F9D019CA92DA38F680BC3F425E8F911E35142D4A5E222AA4273F7C7C9F59F
6E3E65AF491AF242B5A9DD2C4CF8485318D8E38E6B5976394*/
  Cpy_U32(out, in,32);
    AES_Parm.in = out;
    AES_Parm.key = key;
    AES_Parm.iv = iv;
    AES_Parm.out = out;

    AES_Parm.keyWordLen = 4;
    AES_Parm.inWordLen = 32;

    AES_Parm.Mode = AES_ECB;
    AES_Parm.En_De = AES_ENC;
    ret =AES_Init(&AES_Parm);
    ret = AES_Crypto(&AES_Parm);
    AES_Close();

    if(ret!= AES_Crypto_OK)
    {
        flag1=0x5A5A5A5A;
    }
    else
```

```
{
    if(Cmp_U32(AES_ECB_EN, 32, out, 32))
    {
        flag1=0x5A5A5A5A;
    }
    else
    {
        flag1=0;
    }
}
Cpy_U32(out, in,32);
AES_Parm.En_De = AES_DEC;
ret =AES_Init(&AES_Parm);
ret = AES_Crypto(&AES_Parm);
AES_Close();
if(ret!= AES_Crypto_OK)
{
    flag2=0x5A5A5A5A;
}
else
{

    if(Cmp_U32(AES_ECB_DE, 32, out, 32))
    {
        flag2=0x5A5A5A5A;
     }
    else
    {
            flag2=0;
        }
    }
    //CBC
Cpy_U32(out, in,32);
    AES_Parm.Mode = AES_CBC;
    AES_Parm.En_De = AES_ENC;
    ret =AES_Init(&AES_Parm);
    ret = AES_Crypto(&AES_Parm);
    AES_Close();
    if(ret!= AES_Crypto_OK)
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        if(Cmp_U32(AES_CBC_EN, 32, out, 32))
        {
             flag3=0x5A5A5A5A;
        }
        else
        {
            flag3=0;
        }
    }
  Cpy_U32(out, in,32);
    AES_Parm.En_De = AES_DEC;
    ret =AES_Init(&AES_Parm);
  ret = AES_Crypto(&AES_Parm);
  AES_Close();
  if(ret!= AES_Crypto_OK)
  {
      flag4=0x5A5A5A5A;
  }
  else
  {
      if(Cmp_U32(AES_CBC_DE, 32, out, 32))
      {
          flag4=0x5A5A5A5A;
      }
      else
      {
          flag4=0;
      }
  }
  //CTR
Cpy_U32(out, in,32);
  AES_Parm.Mode = AES_CTR;
  AES_Parm.En_De = AES_ENC;
  ret =AES_Init(&AES_Parm);
  ret = AES_Crypto(&AES_Parm);
  AES_Close();
  if(ret!= AES_Crypto_OK)
  {
      flag5=0x5A5A5A5A;
  }
  else
  {
      if(Cmp_U32(AES_CTR_EN, 32, out, 32))
      {
           flag5=0x5A5A5A5A;
      }
      else
      {
          flag5=0;
      }
  }
Cpy_U32(out, AES_CTR_EN,32);
  AES_Parm.En_De = AES_DEC;
  ret =AES_Init(&AES_Parm);
  ret = AES_Crypto(&AES_Parm);
  AES_Close();
  if(ret!= AES_Crypto_OK)
  {
     flag6=0x5A5A5A5A;
  }
  else
  {
      if(Cmp_U32(AES_CTR_DE, 32, out, 32))
      {
          flag6=0x5A5A5A5A;
      }
      else
      {
```

```
                flag6=0;
            }
        }

    if (flag1|flag2|flag3|flag4|flag5|flag6)
        {
            return 0x5A5A5A5A;
        }
        else
        {
            return 0;
        }

}



u32 AES_192_test()
{
        u32 flag1,flag2,flag3,flag4,flag5,flag6,ret=0;
        AES_PARM AES_Parm={0};

                u32
in[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,0xBA43D804,0xDE77B9E1,0xE1A330
77,0xE3AEA215,

        0x2670CBEB,0x160CA5C2,0x86808BEA,0x3D7A9E73,0xB16E68A0,0x12E5BF98,0x8A18EC5F,
0xC4BD0D05,
        0xAB21B81D,0x7477E171,0xDE6FFEF4,0xB80B68F8,0xA4AF05A1,0x1C77249A,0xB2CCA806,0x9C3A69BA,

        0x6F7CD7A9,0x2BD9E19F,0x78B41533,0x2F5E08F7,0x1C2EF8F1,0x03D4B04F,0xE0EAAC56,0x73CC7E9C};
            u32 key[6]={0xA1148977,0xCFA42A1F,0x9D983F36,0x521C1313,0xDAD2CB6F,0xC6254819};

            u32 iv[4]={0xFCAA7077,0x44DB6BB5,0xDC74178D,0xA91A44D6};
            u32 out[32];

            u32 AES_ECB_EN[32]={0x9FCB396D,0xF9A6B55C,0x4CCE7669,0x917CAF2F,0x71F8907D,0xC6893936,0x5ABA1DFB,0xA933FF81,

        0xBD33847F,0x0F1B2F6C,0x1B4AACA7,0xE555E2EE,0x0CBD4683,0x76ECD138,0x7BFE81E8,0xE05FE788,

        0xAF688124,0xED29ACF2,0xCE424458,0x8E304A1C,0xE5A21E6C,0x3C7D433A,0x32DC028D,0x697F9624,

        0xB451070E,0xF82A4488,0x33D99F4C,0x7FBBCC3E,0x8BB01E57,0x0C1EE01B,0x6D96FF7F,0xDEC84BD8};
            u32 AES_ECB_DE[32]={0x41F29D18,0x13C52105,0xB24DBDDD,0x46B6BAB9,0x95F63F1A,0x28B24F73,0xAA774293,0xA086E548,
            0xD446667D,0xF8D67CCE,0x7AC5BD02,0xE43EE791,0x25B857B4,0x30A3D7FB,0x8DB4C416,0xAE6B0B0C,

            0x0F7E89E1,0xBA900B96,0x516EC69B,0xBED1D082,0x3590FD32,0x878C5EE5,0x91B71430,0x6A005A7F,

            0x0627EF04,0x28D96A77,0xF8DCDCFC,0x790D0304,0x02149E37,0xDC8E518D,0x80D75D77,0x80670408};

                u32 AES_CBC_EN[32]={0xE5682F2E,0x07A087E9,0x37D60ED6,0x41262C81,0xD69A23B5,0x1800A3FD,0xAC50301D,0xB12F3C5E,

            0x568A1F62,0xC1057524,0x7E7D09BC,0x26F42541,0x5C2FB09B,0x12C68EFC,0xE03B2AF8,0x6E2C9934,

            0xD805445F,0x3876A6E4,0xCA85688F,0xD1116501,0x2DE18902,0xCBFDE9B2,0x57911796,0x0719A673,

            0x3915B680,0x3B760C23,0x23F715DE,0x6D3425B9,0x9C339EF5,0x6C91D7B0,0x050E91DA,0x286AB477};
                u32 AES_CBC_DE[32]={0xBD58ED6F,0x571E4AB0,0x6E39AA50,0xEFACFE6F,0xCFB4F836,0x21432C5A,0x43CA36B8,0x148505B7,

            0x6E05BE79,0x26A1C52F,0x9B668D75,0x07904584,0x03C89C5F,0x26AF7239,0x0B344FFC,0x9311957F,
            0xBE10E141,0xA875B40E,0xDB762AC4,0x7A6CDD87,0x9EB1452F,0xF3FBBF94,0x4FD8EAC4,0xD20B3287,

            0xA288EAA5,0x34AE4EED,0x4A1074FA,0xE5376ABE,0x6D68499E,0xF757B012,0xF8634844,0xAF390CFF};

                    u32 AES_CTR_EN[32]={0xF4EB3E15,0xCEC90E4B,0x1708E770,0x6A1297BB,0x045A69FD,0x7FC870A7,0x56BE6A22,0x5A912CEA,

    0xC22E6811,0x37177967,0x68D08A6A,0xCECA04AE,0x30EA7217,0x16992F79,0xF0DD4DAD,0x4710126B,0xCC06BD7F,

    0x03093EE5,0x596D2B9B,0xD9844F7C,0x130D4E24,0xD6C87ABF,0xE1745614,0xEF260225,0x0F90C354,0x7557E159,

    0x4CBC3789,0xDB0552F8,0x28F27315,0x046363A6,0xAF1F0089,0x29AC2CC1};

                    u32 AES_CTR_DE[32]={0x5A42C72C,0x09F16329,0xE9BD742B,0xB403E0FF,0xBA43D804,0xDE77B9E1,0xE1A33077,0xE3AEA215,

            0x2670CBEB,0x160CA5C2,0x86808BEA,0x3D7A9E73,0xB16E68A0,0x12E5BF98,0x8A18EC5F,0xC4BD0D05,

            0xAB21B81D,0x7477E171,0xDE6FFEF4,0xB80B68F8,0xA4AF05A1,0x1C77249A,0xB2CCA806,0x9C3A69BA,
        0x6F7CD7A9,0x2BD9E19F,0x78B41533,0x2F5E08F7,0x1C2EF8F1,0x03D4B04F,0xE0EAAC56,0x73CC7E9C};

            AES_Parm.in = in;
            AES_Parm.key = key;
            AES_Parm.iv = iv;
            AES_Parm.out = out;

            AES_Parm.keyWordLen = 6;
            AES_Parm.inWordLen = 32;

            AES_Parm.Mode = AES_ECB;
            AES_Parm.En_De = AES_ENC;
            ret =AES_Init(&AES_Parm);
            ret =AES_Crypto(&AES_Parm);
            AES_Close();

            if(Cmp_U32(AES_ECB_EN, 32, out, 32))
            {
                    flag1=0x5A5A5A5A;
            }
            else
            {
                    flag1=0;
            }

            AES_Parm.En_De = AES_DEC;
            ret =AES_Init(&AES_Parm);
            ret =AES_Crypto(&AES_Parm);
            AES_Close();



            if(Cmp_U32(AES_ECB_DE, 32, out, 32))
            {
                    flag2=0x5A5A5A5A;
            }
            else
            {
                    flag2=0;
            }
        //cbc
             AES_Parm.Mode = AES_CBC;
             AES_Parm.En_De = AES_ENC;
            ret =AES_Init(&AES_Parm);
            ret =AES_Crypto(&AES_Parm);
            AES_Close();

            if(Cmp_U32(AES_CBC_EN, 32, out, 32))
            {
                    flag3=0x5A5A5A5A;
            }
            else
            {
                    flag3=0;
            }
            AES_Parm.En_De = AES_DEC;
            ret =AES_Init(&AES_Parm);
            ret =AES_Crypto(&AES_Parm);
            AES_Close();

            if(Cmp_U32(AES_CBC_DE, 32, out, 32))
            {
                    flag4=0x5A5A5A5A;
            }
            else
            {
                    flag4=0;
            }

        //ctr
            AES_Parm.Mode = AES_CTR;
            AES_Parm.En_De = AES_ENC;
            ret =AES_Init(&AES_Parm);
            ret =AES_Crypto(&AES_Parm);
            AES_Close();

            if(Cmp_U32(AES_CTR_EN, 32, out, 32))
            {
                    flag5=0x5A5A5A5A;
            }
            else
            {
                    flag5=0;
```

```
        }
    AES_Parm.in = AES_CTR_EN;
        AES_Parm.En_De = AES_DEC;
        ret =AES_Init(&AES_Parm);
        ret =AES_Crypto(&AES_Parm);
        AES_Close();

        if(Cmp_U32(AES_CTR_DE, 32, out, 32))
        {
                flag6=0x5A5A5A5A;
        }
        else
        {
                flag6=0;
        }



    if (flag1|flag2|flag3|flag4|flag5|flag6)
      {
            return 0x5A5A5A5A;
      }
      else
      {
            return 0;
      }
}

u32 AES_256_test()
```

```
{
    u32 flag1,flag2,flag3,flag4,flag5,flag6,ret=0;
    AES_PARM AES_Parm={0};

    u32 in[32]={0x86DF711D,0xB9C4122D,0x13368B2D,0x53A5CF4F,0xBDFFAA2C,0xB4D4B3C0,0x8BB97CB6,0x99EA0BE6,

    0x8B338E1D,0xFE104A1C,0x4E13D5E3,0xA886852F,0x67522841,0x9D1FF5E1,0xEFBDC3A3,0xA7C27969,

    0x0475C629,0xD4EB12F0,0x4570B427,0xF9296516,0x58F7F4A6,0x2A9D3C6B,0x652654E1,0x438105F6,

    0x986F81C9,0x639F51B2,0xA3169082,0x6CD5570C,0x39B678E4,0x84986F66,0x94BB95FA,0x976D9797};
    u32 key[8]={0xB2591B82,0xD25676DB,0x2546F076,0xC8D01753,0xB4A620E7,0x4AADD91D,0x2E5EDF9B,0x596C1146};

    u32 iv[4]={0xF0E72786,0xD272F169,0x0ECED17B,0x29D34319};
    u32 out[32];

    u32 AES_ECB_EN[32]={0x5766DACC,0x50DBB1F9,0x58720E73,0x2182AA3E,0x7D5A6D4D,0xA07EF43D,0x5A533E1E,0x34816CF3,
        0xBA23F9CD,0x99A7BD14,0x6789D933,0xD14B2F0D,0xAF53E19E,0xB88DA31F,0xEFBE0472,0x03F077B1,

        0x4489E477,0x97161707,0x6C24CB62,0x0FF361DC,0x60BBD2CF,0xEB7AB0C1,0xFA3421E5,0x2F5DB80E,

        0x2D61A7CD,0x22988E98,0x51B195AF,0x22C8A4C0,0x7F8E90C3,0x6690789A,0x48AF0FAF,0xAC16F7A6};
        u32 AES_ECB_DE[32]={0x0ADBDA93,0x93C512ED,0x6A99A60B,0x0A1841B5,0x135E685D,0xB9ADC987,0x6262573F,0x9090A7D3,

        0x2B7DDAA3,0x7370FB9D,0xE7E739C6,0xCA013CA6,0x3509E08F,0x74A21641,0x3D2C9527,0xF8DF90F0,

        0xED8209E9,0x9DD57975,0x0A506603,0x7C2EFD3B,0x0937237E,0x2828BAAF,0x245E9D40,0xF3BB882A,

        0x66E82B24,0xF3E778E7,0x386802D1,0xD74C7057,0xEF8525C8,0x1EB7AA48,0x362EACDD,0x8AA0F286};

        u32 AES_CBC_EN[32]={0x39AD6F3A,0xF8E3E1DD,0x2209A14B,0x241642CC,0x83FA4820,0xD82816B3,0xEF66B17A,0xB5B49FCC,

        0xA7540FD7,0xCC11801C,0xC6126D93,0x8E6C259A,0x626135EB,0x3FEA411B,0x45FF91A3,0x1B91B51A,
        0x9169DD4C,0x2F42A1E6,0x4299E687,0xEB9FBAA4,0x3B667902,0xDCB4117A,0x45B78A05,0x5FECBFA7,

        0x54C54A81,0xBDF538B1,0xF2D5804D,0x568910A8,0x41655B32,0xD47D533B,0x5A82D212,0x63C07B46};
        u32
AES_CBC_DE[32]={0xFA3CFD15,0x41B7E384,0x64577770,0x23CB02AC,0x95811940,0x0069DBAA,0x7154DC12,0xC335689C,

        0x9682708F,0xC7A4485D,0x6C5E4570,0x53EB3740,0xBE3A6E92,0x8AB25C5D,0x733F40C4,0x505915DF,

        0x8AD021A8,0x00CA8C94,0xE5EDA5A0,0xDBEC8452,0x0D42E557,0xFCC3A85F,0x612E2967,0x0A92ED3C,

        0x3E1FDF82,0xD97A448C,0x5D4E5630,0x94CD75A1,0x77EAA401,0x7D28FBFA,0x95383C5F,0xE675A58A};
        u32
AES_CTR_EN[32]={0x85F1DD33,0xAE808F2F,0x26A40960,0xB2020DF8,0xB6C2006E,0xA22A35F6,0x33BB584A,0xBFEA7F68,

        0x73E54E78,0xF3EB0368,0x80816676,0x6109DE39,0xE0001920,0x8D2B18B8,0x0E46A012,0xE43F1DD1,0x3CA4BC36,

        0xD5101452,0x83020170,0x4B752F62,0x3D27A004,0x3C18B5DB,0x99DA9032,0xEA59B340,0x79BBD087,0x2EF8CB3D,
    0xDC32D3CA,0x30F577EA,0x56774C66,0xC33DA1F8,0x0288B1D6,0x091C9666};
    u32 AES_CTR_DE[32]={0x86DF711D,0xB9C4122D,0x13368B2D,0x53A5CF4F,0xBDFFAA2C,0xB4D4B3C0,0x8BB97CB6,0x99EA0BE6,

    0x8B338E1D,0xFE104A1C,0x4E13D5E3,0xA886852F,0x67522841,0x9D1FF5E1,0xEFBDC3A3,0xA7C27969,

    0x0475C629,0xD4EB12F0,0x4570B427,0xF9296516,0x58F7F4A6,0x2A9D3C6B,0x652654E1,0x438105F6,

    0x986F81C9,0x639F51B2,0xA3169082,0x6CD5570C,0x39B678E4,0x84986F66,0x94BB95FA,0x976D9797};


    AES_Parm.in = in;
    AES_Parm.key = key;
    AES_Parm.iv = iv;
    AES_Parm.out = out;

    AES_Parm.keyWordLen = 8;
    AES_Parm.inWordLen = 32;

    AES_Parm.Mode = AES_ECB;
    AES_Parm.En_De = AES_ENC;
    ret =AES_Init(&AES_Parm);
    ret =AES_Crypto(&AES_Parm);
    AES_Close();

    if(Cmp_U32(AES_ECB_EN, 32, out, 32))
    {
        flag1=0x5A5A5A5A;
    }
    else
    {
        flag1=0;
    }

    AES_Parm.En_De = AES_DEC;
    ret =AES_Init(&AES_Parm);
    ret =AES_Crypto(&AES_Parm);
    AES_Close();


    if(Cmp_U32(AES_ECB_DE, 32, out, 32))
    {
        flag2=0x5A5A5A5A;
    }
    else
    {
        flag2=0;
    }
//CBC
    AES_Parm.Mode = AES_CBC;
    AES_Parm.En_De = AES_ENC;
    ret =AES_Init(&AES_Parm);
    ret =AES_Crypto(&AES_Parm);
    AES_Close();

    if(Cmp_U32(AES_CBC_EN, 32, out, 32))
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        flag3=0;
    }

    AES_Parm.En_De = AES_DEC;
    ret =AES_Init(&AES_Parm);
    ret =AES_Crypto(&AES_Parm);
    AES_Close();

    if(Cmp_U32(AES_CBC_DE, 32, out, 32))
    {
        flag4=0x5A5A5A5A;
    }
    else
    {
        flag4=0;
    }
//CTR
    AES_Parm.Mode = AES_CTR;
```

```
  AES_Parm.En_De = AES_ENC;
  ret =AES_Init(&AES_Parm);
  ret =AES_Crypto(&AES_Parm);
  AES_Close();

  if(Cmp_U32(AES_CTR_EN, 32, out, 32))
  {
      flag5=0x5A5A5A5A;
  }
  else
  {
      flag5=0;
  }
AES_Parm.in = AES_CTR_EN;
  AES_Parm.En_De = AES_DEC;
  ret =AES_Init(&AES_Parm);
  ret =AES_Crypto(&AES_Parm);
  AES_Close();

  if(Cmp_U32(AES_CTR_DE, 32, out, 32))
  {
      flag6=0x5A5A5A5A;
  }
  else
  {
      flag6=0;
  }
```

```
    if (flag1|flag2|flag3|flag4|flag5|flag6)
    {
        return 0x5A5A5A5A;
    }
    else
    {
        return 0;
    }
}
```

ii.附录二 SM4算法库函数调用例程

```
u32 SM4_test(void)
{
    u32 flag1,flag2,flag3,flag4;
    u32 ret;
    SM4_PARM SM4_Parm={0};
    /*若需要修改测试实例，当参数的真实值为“0x0102030405060708”时，由于 u32 数据是字节小端序存储，在对以上参数进行初始化赋值时，请输入“0x04030201,0x08070605”.若无特殊说明，本例程参数都以这种方式设置*/
    u32 in1[32]={
        0x4B551C70,0xD54DA600,0xBAA2CA7F,0x0ABA6CD8,0x97BC9D7D,0xAD650748,
        0x0590F143,0x7288FD0F,0x9EDF1005,0xB7D4A607,0x8ED480C9,0x34FD4C59,
        0x97C9286E,0xD0A23857,0x1ABE2026,0x6163578A,0xF5FBAFB4,0x72DB71B7,
        0x21217431,0xF8BE4ECA,0xB73D1018,0xACD37812,0x3FF19EE7,0x4C9575BE,
        0xF1FB289E,0x33694113,0x8EC5BB10,0x3B1DFF5F,0xA9D6A5A5,0xB98D90C8,
        0x91AB4E89,0x804343FD
    };
    u32 key1[4]={0x84853E30,0xB3D3154D,0x9A887F49,0xDC65910A};
    u32 iv1[4]={0x2FA6B65A,0x1D0EC205,0xB90B8620,0x42E74F58};
    u32 out[32];
    u32 SM4_ECB_EN[32]={0xD61A389C,0xE136A0AD,0xBD626B7E,0x4277F173,0xAF3E5E82,
    0x876D84DF,0x7A065B7B,0x1CBBFFA8,0xC57C31DC,0x5BD86AFC,
    0x0825EAEF,0x600162A4,0x3E4787AC,0x58B32579,0x3A9135BF,
    0xB806A17C,0x9854F4C4,0x065CD28F,0x68FDF21F,0x9CA62C4C,
    0x5B2FA76E,0xEC693A2B,0xF028ADF6,0xFAA2ED18,0x6395B4B1,
    0x7A9B0069,0x9D55E04C,0xA5CDC23F,0x7FC56C92,0x89F199A1,
    0xF228D9E1,0xD705050A};
    /*SM4_ECB_EN=0x9C381AD6ADA036E17E6B62BD73F17742825E3EAFDF846D877B5B067A
A8FFBB1CDC317CC5FC6AD85BEFEA2508A4620160AC87473E7925B358BF35913A7CA106B8C4F
454988FD25C061FF2FD684C2CA69C6EA72F5B2B3A69ECF6AD28F018EDA2FAB1B4956369009B
7A4CE0559D3FC2CDA5926CC57FA199F189E1D928F20A0505D7*/

    u32 SM4_ECB_DE[32]={0x3107DFA0,0xC1EE3D0A,0x9025F9D5,0x90ACC081,0x7A72F90A,
    0x6481F1CE,0x76DF5450,0xCD262ACF,0xCE8E3C3B,0x208B7390,
    0xC9F8F526,0x1A73FFCC,0x0AB6E26F,0xA02B544A,0x760CD602,
    0x6D250CA4,0x2477FF67,0x44CBC39E,0x84ECF5CC,0x7DF30644,
    0x8746D41C,0xCB42B9EC,0xE975598C,0x28756C41,0x64C3C870,
    0x9EA8CBB3,0xBA2FA98E,0x1B10BA7B,0x1C50E8A0,0x1EE697FD,
    0xA4E2DDD5,0xBB29D912};
    /*SM4_ECB_DE=0xA0DF07310A3DEEC1D5F9259081C0AC900AF9727ACEF181645054DF76C
F2A26CD3B3C8ECE90738B2026F5F8C9CCFF731A6FE2B60A4A542BA002D60C76A40C256D67FF
77249EC3CB44CCF5EC844406F37D1CD44687ECB942CB8C5975E9416C752870C8C364B3CBA89E
8EA92FBA7BBA101BA0E8501CFD97E61ED5DDE2A412D929BB*/

    u32 SM4_CBC_EN[32]={0x304E1C3C,0x10DA649D,0x5EBCB5BE,0x2964AD84,0x18599756,
    0x2106AAD2,0x84364B24,0x57A9E62D,0xD160B03B,0x58293A74,
    0xEE57389F,0x398E69C2,0x63FD0959,0x5B4584FD,0x4DA6E8BE,
    0x578E4501,0x74B0159B,0x570E8604,0x38E2DB49,0xE028387E,
    0xCDDE4984,0x6B717E9F,0xE516D698,0x6520025E,0xC8D187A7,
    0x6E08373F,0xC3472666,0x654A0D41,0x7F363B95,0xAD8EB5D2,
    0x01F0F12A,0x8169D65A};
    /*SM4_CBC_EN=0x3C1C4E309D64DA10BEB5BC5E84AD642956975918D2AA0621244B36842
DE6A9573BB060D1743A29589F3857EEC2698E395909FD63FD84455BBEE8A64D01458E579B15B0
```

```
7404860E5749DBE2387E3828E08449DECD9F7E716B98D616E55E022065A787D1C83F37086E6626
47C3410D4A65953B367FD2B58EAD2AF1F0015AD66981*/

       u32 SM4_CBC_DE[32]={0x1EA169FA,0xDCE0FF0F,0x292E7FF5,0xD24B8FD9,0x3127E57A,
       0xB1CC57CE,0xCC7D9E2F,0xC79C4617,0x5932A146,0x8DEE74D8,
       0xCC680465,0x68FB02C3,0x9469F26A,0x17FFF24D,0xF8D856CB,
       0x59D840FD,0xB3BED709,0x9469FBC9,0x9E52D5EA,0x1C9051CE,
       0x72BD7BA8,0xB999C85B,0xC8542DBD,0xD0CB228B,0xD3FED868,
       0x327BB3A1,0x85DE3769,0x5785CFC5,0xEDABC03E,0x2D8FD6EE,
       0x2A2766C5,0x8034264D};
       /*SM4_CBC_DE=0xFA69A11E0FFFE0DCF57F2E29D98F4BD27AE52731CE57CCB12F9E7DCC
17469CC746A13259D874EE8D650468CCC302FB686AF269944DF2FF17CB56D8F8FD40D85909D7
BEB3C9FB6994EAD5529ECE51901CA87BBD725BC899B9BD2D54C88B22CBD068D8FED3A1B37
B326937DE85C5CF85573EC0ABEDEED68F2DC566272A4D263480*/
        Cpy_U32(out, in1,32);
       SM4_Parm.in = out;
       SM4_Parm.key = key1;
       SM4_Parm.out = out;
       SM4_Parm.inWordLen = 32;
       SM4_Parm.workingMode = SM4_ECB;
       SM4_Parm.EnDeMode = SM4_ENC;
        ret=SM4_Init(&SM4_Parm);
       ret=(SM4_Crypto(&SM4_Parm));
       SM4_Close();
      if(ret!=SM4_Crypto_OK)
       {
              flag1=0x5A5A5A5A;
       }
```

```
 else
 {
        if(Cmp_U32(SM4_ECB_EN,32, out,32))
        {
                flag1=0x5A5A5A5A;
        }
       else
        {
                flag1=0;
        }
 }
Cpy_U32(out, in1,32);
 SM4_Parm.EnDeMode = SM4_DEC;
ret=SM4_Init(&SM4_Parm);
 ret=(SM4_Crypto(&SM4_Parm));
 SM4_Close();
 if(ret!=SM4_Crypto_OK)
 {
        flag2=0x5A5A5A5A;
 }
 else
 {

        if(Cmp_U32(SM4_ECB_DE,32, out,32))
        {
                flag2=0x5A5A5A5A;
        }
          else
```

```
    {
        flag2=0;
    }
}
Cpy_U32(out, in1,32);
SM4_Parm.iv = iv1;
SM4_Parm.workingMode = SM4_CBC;
SM4_Parm.EnDeMode = SM4_ENC;
 ret=SM4_Init(&SM4_Parm);
ret=(SM4_Crypto(&SM4_Parm));
SM4_Close();
if(ret!=SM4_Crypto_OK)
{
    flag3=0x5A5A5A5A;
}
else
{
    if(Cmp_U32(SM4_CBC_EN,32, out,32))
    {
        flag3=0x5A5A5A5A;
    }
    else
    {
        flag3=0;
    }
}
Cpy_U32(out, in1,32);
SM4_Parm.iv= iv1;
```

```
SM4_Parm.EnDeMode = SM4_DEC;
ret=SM4_Init(&SM4_Parm);
ret=(SM4_Crypto(&SM4_Parm));
SM4_Close();
if(ret!=SM4_Crypto_OK)
{
    flag4=0x5A5A5A5A;
}
else
{

    if(Cmp_U32(SM4_CBC_DE,32, out,32))
     {
        flag4=0x5A5A5A5A;
    }
    else
    {
        flag4=0;
    }
}

if (flag1|flag2|flag3|flag4)
{
    return 0x5A5A5A5A;
}
else
{
    return 0;
    }
}
```

iii.附录三 RNG算法库调用例程

```
#define POKER_RAND_BYTE 40 //320bit
u32 TrueRand_Poker_Test(void)
{
    u16 count[16] = {0};
    u32 sum = 0;
    u8   rand[POKER_RAND_BYTE];
    u8 i, j, k, tmp;

    GetTrueRand_U32((u32*)rand, POKER_RAND_BYTE>>2);
    //GetTrueRand_U8(rand, POKER_RAND_BYTE);
    //GetPseudoRand_U32((u32*)rand,POKER_RAND_BYTE>>2);
    for(j = 0; j < POKER_RAND_BYTE; j++)
    {
        for(k = 0; k < 2; k++)
        {
            (k == 1) ? tmp = (rand[j] >> 4) : (tmp = (rand[j] & 0x0F));
            for(i = 0; i < 16; i++)
            {
                if(tmp==i) count[i]++;
            }
        }
    }
    for(i = 0; i < 16; i++)
    {
        sum += ((u32)count[i]) * count[i];
    }

    if(405 < sum && sum < 687)
        return 0;
    else
        return 1;
}
u32 PseudoRand_Poker_Test(void)
{
    u16 count[16] = {0};
    u32 sum = 0;
    u8   rand[POKER_RAND_BYTE];
    u8 i, j, k, tmp;

    //GetTrueRand_U32((u32*)rand, POKER_RAND_BYTE>>2);
    //GetTrueRand_U8(rand, POKER_RAND_BYTE);
    GetPseudoRand_U32((u32*)rand,POKER_RAND_BYTE>>2,NULL);
    for(j = 0; j < POKER_RAND_BYTE; j++)
    {
        for(k = 0; k < 2; k++)
        {
            (k == 1) ? tmp = (rand[j] >> 4) : (tmp = (rand[j] & 0x0F));
            for(i = 0; i < 16; i++)
            {
                if(tmp==i) count[i]++;
            }
        }
    }
```

```
        for(i = 0; i < 16; i++)
        {
                sum += ((u32)count[i]) * count[i];
        }

        if(405 < sum && sum < 687)
                return 0;
        else
                return 1;
}
```